Computing for Economists

Derivative-Free and & Constrained Optimization

Toren Fronsdal and Ruby Zhang

Harvard University

Introduction

We will discuss two broad classes of optimization problems:

- Derivative-free optimization methods (unconstrained)
- Constrained optimization (both derivative-based and derivative-free)

The heart of constrained optimization is finding ways to break down the problem into successive unconstrained optimization problems

Resources

Most of this lecture draws from the following sources:

- Ken Judd Course (Lectures 4 and 6)
- Lluís Alsedà Notes
- Jesús Fernández-Villaverde Course
- Shunsuke Tsuda Lecture
- Jason Debacker Course

Table of Contents

Optimization Methods

We will discuss the following numerical optimization methods:

- Derivative-Free Methods
  - Grid Search (Brute force)
  - Nelder-Mead (Downhill Simplex Method)
  - One-Dimensional Methods (used as subroutine)
- Constrained Optimization
  - Lagrange Multipliers & KKT Conditions
    - Sequential Quadratic Programming (SQP)
  - Penalty & Barrier Functions
    - Exterior and interior point methods
  - Linear Programming (Special Case)
    - Constrained Optimization BY Linear Approximation (COBYLA)

Derivative-Free Optimization

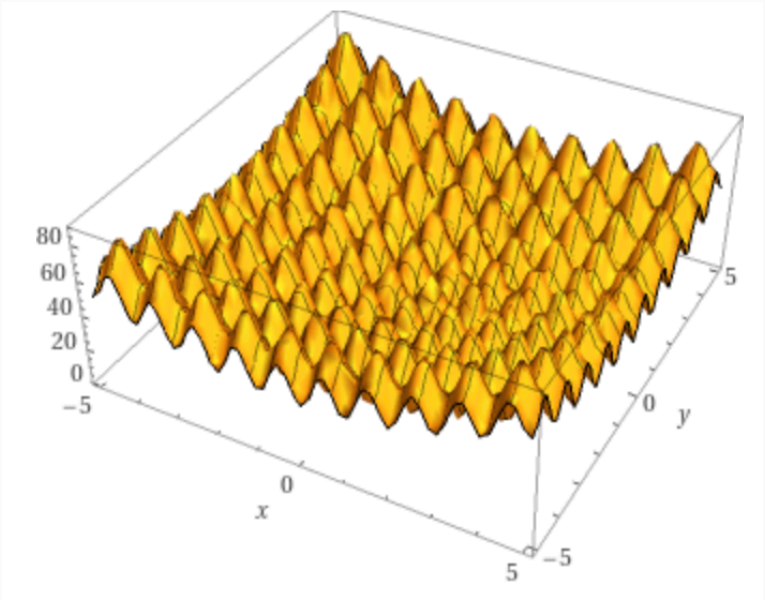
Why Derivative-Free Methods?

Derivative-based methods converge *fast* locally but they need a *derivative*. When can this go wrong?

- Derivative is unavailable: discontinuity, kinks, etc.
- Derivative is computationally intensive
  - Numerical differentiation for each derivative calculation

Example: Rastrigin function with a kink

$$f(x, y) = 20 + x^2 - 10cos(2\pi x) + y^2 - 10cos(2\pi y) + |x - y|$$



Example: Modified Rastrigin Function

The true global minimum is $(0, 0)$, however the kink makes it hard:

```python
import numpy as np
from scipy.optimize import minimize

# Define the objective function
def objective(X):
    x, y = X
    return 20 + (x**2 - 10 * np.cos(2 * np.pi * x))
            + (y**2 - 10 * np.cos(2 * np.pi * y)) + np.abs(x - y)

# Initial guess
x0 = [0.1, 0.1]

# Call the optimizer (Nelder-Mead)
result_free = minimize(objective, x0, method='Nelder-Mead')
result_deriv = minimize(objective, x0, method='BFGS')
```

```
Optimal value of x (derivative-free): -0.0001
Minimum value of the function (derivative-free): 0.0000

Optimal value of x (derivative-based): -0.0019
Minimum value of the function (derivative-based): 0.0015
```

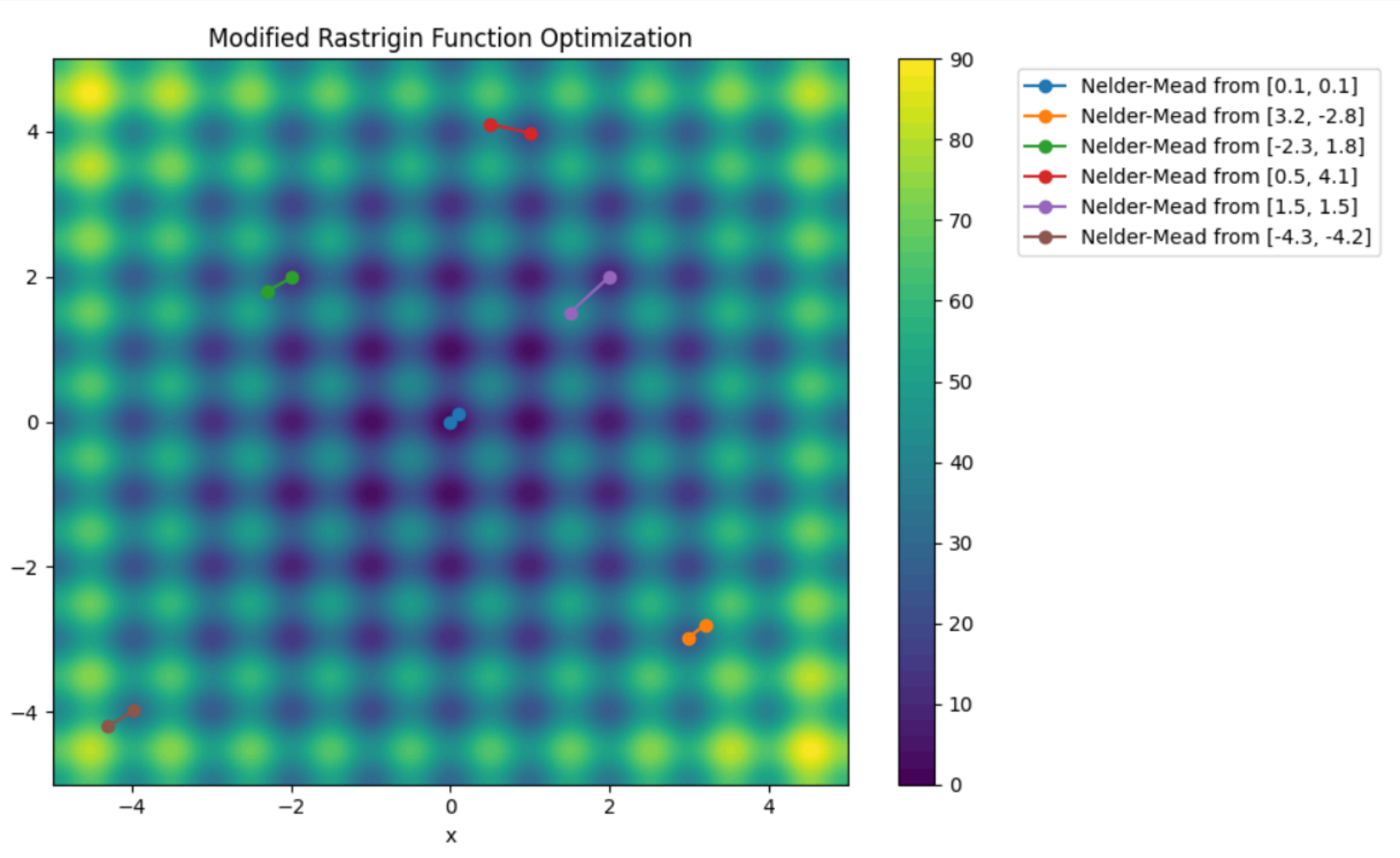
Nelder-Mead vs. BFGS for Close Initial Guess

Note: This is a hard function – even with Nelder-Mead, the initial point matters a lot!

SLSQP (discussed later) performs quite well.

Example: Modified Rastrigin Function

Nelder-Mead with different starting values



Brute Force: Grid Search

Question: Derivative-based solutions work great *locally* – how do we know where do we start?

Idea: Take the most coarse, brute-force search possible to narrow down the neighborhood: split domain into Cartesian grid and compute value.

- Then fine-tune with other methods

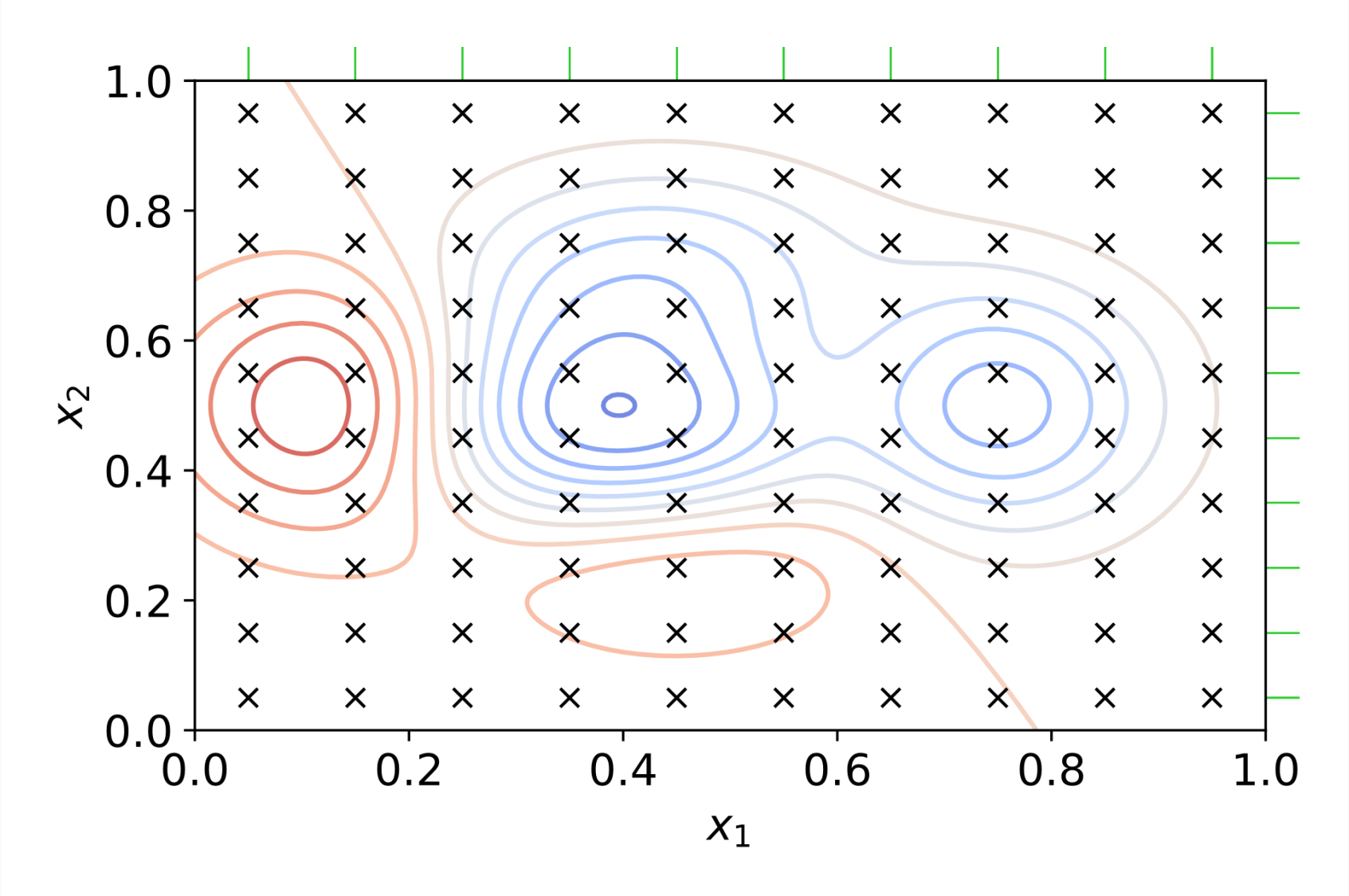
Pros:

- Places no restrictions on the structure of the function, helpful for understanding complex (maybe non-analytical) function
- Easy to parallelize, can isolate several good initial guesses
  - For minimizing (maximizing) strictly convex (concave) functions, any local solution is a global solution

Cons:

- Very slow in higher dimensions (i.e. more than two)

Grid Search



Source: Wikipedia

Downhill Simplex Method (Nelder-Mead)

A *simplex* is a geometric object (with flat sides) that has $n + 1$ vertices in n dimensions (i.e. the simplest possible one).

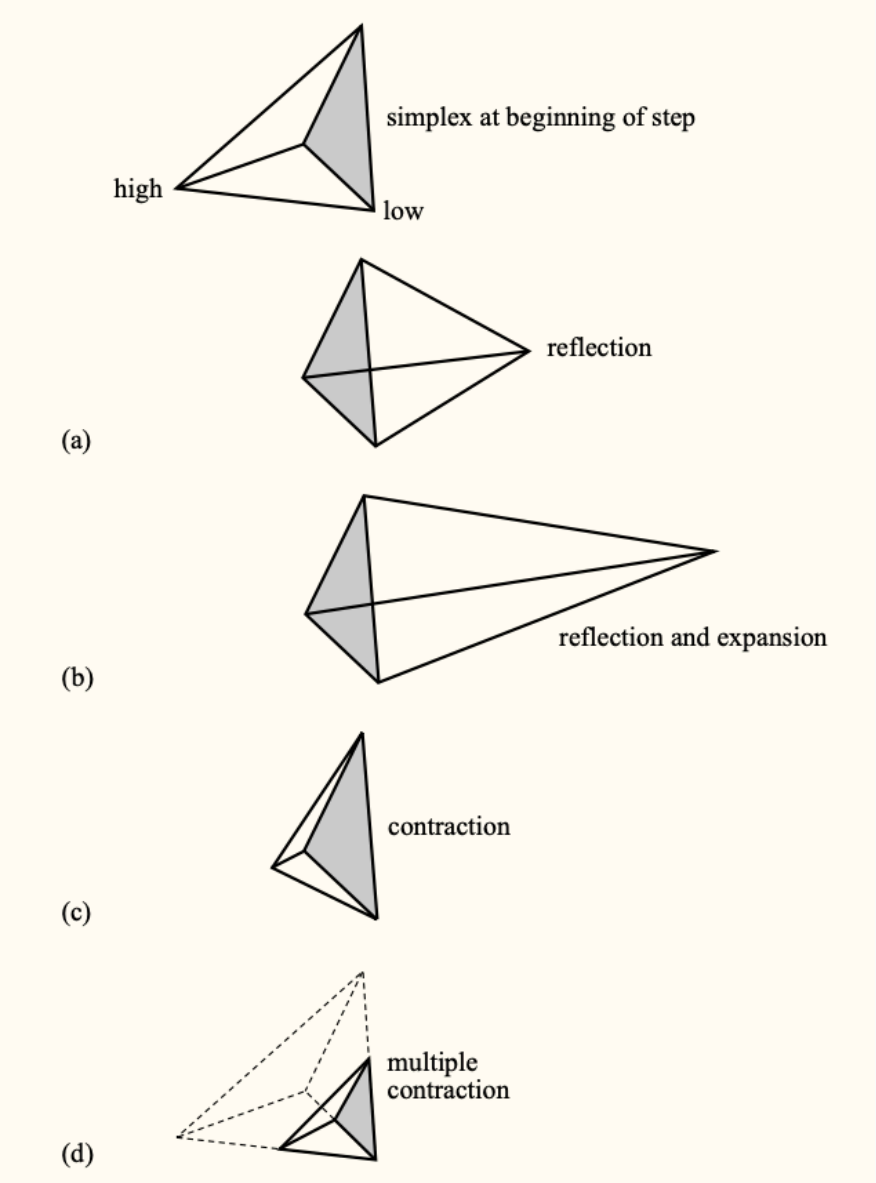
- Line in $\mathbb{R}$, triangle in $\mathbb{R}^2$, tetrahedron in $\mathbb{R}^3$, etc.

Nelder-Mead is faster than grid search, and evaluates a simplex of points each time:

1. Start with a simplex of $N + 1$ points. Order the points from best p_0 (smallest) to worst p_n (largest).
2. Replace the worst point p_n in order using *reflection, expansion, contraction*. If this fails, shrink the simplex towards p_0.[1]
3. N new vertices are initialized in the new simplex. Stop when the decrease in the vector distance moved is smaller than a threshold.

[1] Check out this link for mathematically calculating the exact transformation points.

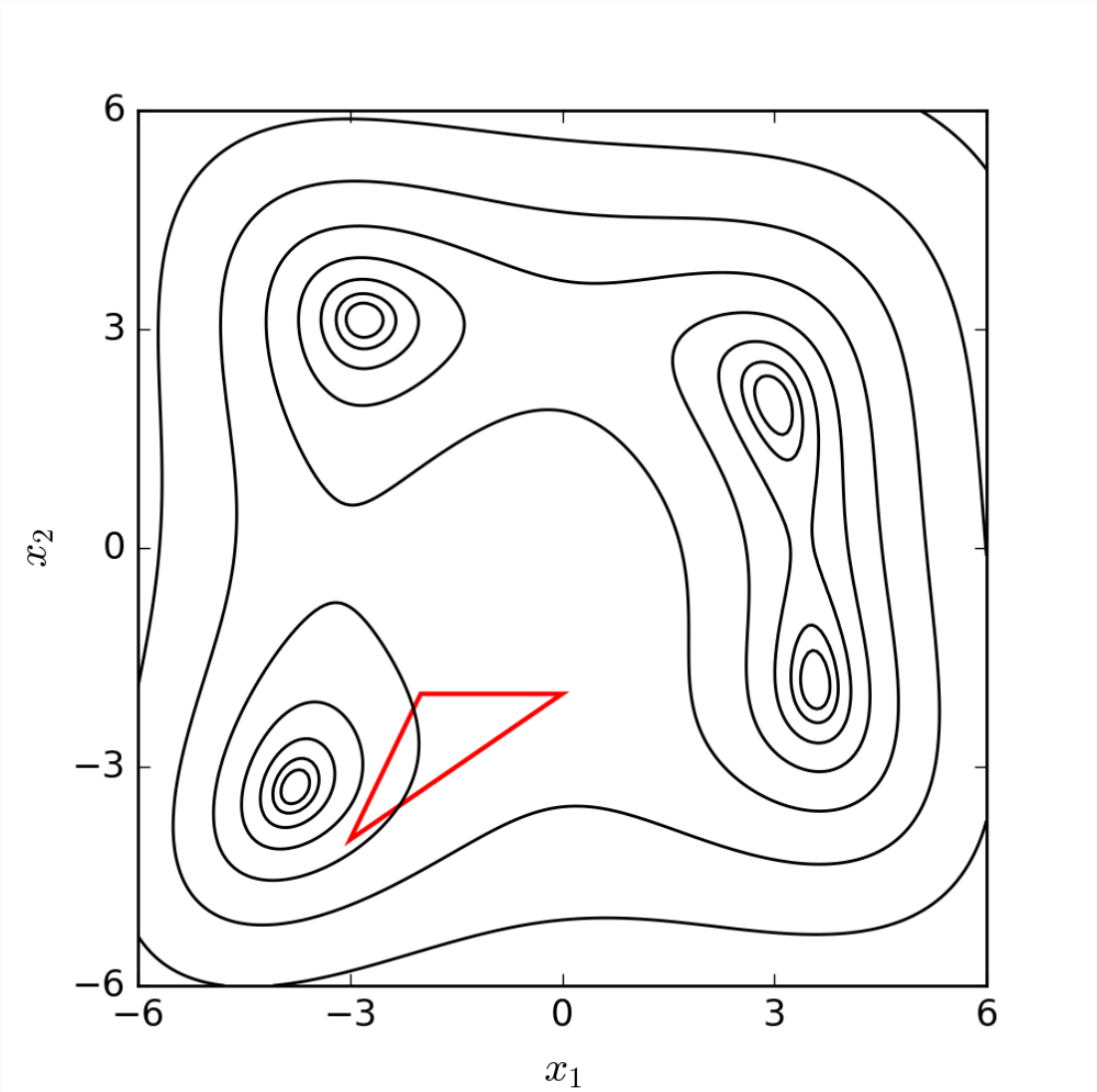
Simplex Transformations



Nelder-Mead is derivative-free and can lead to a reduction in the space of function value to be searched. However, still test with several starting points!

Source: Jesús Fernández-Villaverde

Nelder-Mead in Action



Source: Wikipedia

One-Dimensional Optimization Methods

One-dimensional optimization is often part of a method's subroutine (e.g. gradient descent)

- **Golden Section Search:** robust though slow
  1. For an interval $[a, b]$, calculate the two points given by the golden ratio $\phi = 1.618$ (mathematically more efficient than bisection):

$$x_1 = b - (b - a)/\phi, \qquad x_2 = a + (b - a)/\phi$$

  2. If $f(x_1) < f(x_2)$, then set the new interval as $[a, x_2]$
     If $f(x_1) > f(x_2)$, then set the new interval as $[x_1, b]$
  3. Repeat until the interval is small
- **Brent's Method:** balances speed and robustness
  1. Similar goal of ensuring minimum contained within an interval by combining three methods:
  2. Fit an inverse quadratic curve to a, b, c and find minimum of the curve
  3. If not, use the secant method to update the next point
  4. If not, use bisection method to update the next point

Summary of Unconstrained Optimization

- Grid search is the most brute-force optimization method, however, it is easy to understand
- When available, derivative-based methods are much faster and will converge to a local minimum (hence importance of starting point), quasi-Newtonian methods are the default
- For large-scale problems with smooth functions, employ a broad grid search then quasi-Newtonian method locally
- Nelder-Mead is a direct search method that can be used when the gradient is unavailable
    - Gradient does not exist or computationally expensive to calculate

Constrained Optimization

Constrained Optimization

Many constrained optimization methods “transform” the problem into an unconstrained optimization:

- **Langrange Multipliers / KKT Conditions:** introduce a new variable for each constraint to transform into higher-dimensional unconstrained problem

- **Penalty and Barrier Functions:** add penalty or barrier term to the objective function to keep solution away from infeasible set

- **Linear Programming:** search along the simplex formed by linear constraints and objective function (derivative-free)

Constrained Optimization Algorithms

Under each method, there are specific numerical implementations:

- Langrange Multipliers / KKT Conditions: **Sequential Quadratic Programming (SQP)** is based on quadratic approximation of the Lagrangian
- Penalty and Barrier Functions:
  - **Exterior point methods** are iterative penalty function methods approaching the solution from the infeasible region
  - **Interior point methods** are iterative barrier function methods searching only within the feasible region
- Linear Programming: **Constrained Optimization BY Linear Approximation (COBYLA)** is a derivative-free method solving linear approximations of the problem (based on the simplex)

Lagrangians & Equality Constraints

Given objective $f(x) : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $g(x) : \mathbb{R}^n \to \mathbb{R}^m$,

$$\min_x \quad f(x) \quad s.t. \quad g(x) = 0$$

Transform problem into an $n + m$ dimensional unconstrained optimization by using Lagrange multipliers $\{\lambda_i\}_{i=1}^m$:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

Taking FOC's, we have a set of $n + m$ equality constraints with $n + m$ unknowns to satisfy in order to identify critical points:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial f(x)}{\partial x} + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i(x)}{\partial x} = \mathbf{0}, \qquad \frac{\partial \mathcal{L}}{\partial \lambda} = g(x) = \mathbf{0}$$

This is a system of **nonlinear equations** – use Newtonian methods!

Note this will require calculating the Hessian of the Lagrangian. Quasi-Newton methods such as BFGS avoid this.

Lagrangians & Inequality Constraints

What happens when there are **inequality** constraints $h(x) : \mathbb{R}^n \to \mathbb{R}^k$?

$$\min_x \quad f(x) \quad s.t. \quad g(x) = 0, \quad h(x) \leq 0$$

Let $\{\mu_j\}_{j=1}^k$ be the Lagrange multipliers for the inequality constraints

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{k} \mu_j h_j(x)$$

Karush-Kuhn-Tucker conditions:

$$\frac{\partial f(x)}{\partial x} + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i(x)}{\partial x} + \sum_{j=1}^{k} \mu_j \frac{\partial h_j(x)}{\partial x} = \mathbf{0}$$

$$g(x) = \mathbf{0}, \qquad h(x) \leq \mathbf{0}$$

$$\mu_j = 0 \text{ or } h_j(x) = 0$$

$$\lambda, \mu \geq \mathbf{0}$$

Karush-Kuhn-Tucker (KKT) Conditions

Since $\mu_j = 0$ or $h_j(x) = 0$, this is combined into the **complementary slackness** condition:

$$\mu_j h_j(x) = 0 \ \ \forall \ \ j = \{1, ..., k\}$$

How do we turn this into an equality-constrained problem? One possibility is the **active set method**:

1. Let $\mathcal{I}$ denote the full set of inequality constraints, assume an "active" subset $\mathcal{A} \subset \mathcal{I}$ and solve the problem as a set of nonlinear equations:

$$h_j(x) = 0 \ \ \forall \ \ j \in \mathcal{A}, \qquad \mu_j = 0 \ \ \forall \ \ j \in \mathcal{J} - \mathcal{A}$$

2. Check the solution for all constraints, add violated constraints to the active set, remove slack constraints (i.e. negative Lagrange multipliers) from the active set
3. Repeat until all constraints are satisfied

This is costly for highly nonlinear expressions and if $|\mathcal{I}|$ is large

Sequential Quadratic Programming (SQP)

Idea: Iteratively solve a series of quadratic programming problems which is easier to apply the active set method (Further reading)

- A second-order approximation of the objective function
- First-order approximations of the constraints

Let Δ denote the gradient and $H\mathcal{L}$ the Hessian of the Lagrangian. At each step n, solve for the step size d:

$$\min_d f(x_n) + \Delta f^T(x_n)d + \frac{1}{2}d^T H\mathcal{L}(x_n)d$$

$$\text{where } g(x_n) + \Delta g(x_n)d = 0$$

$$\text{and } h(x_n) + \Delta h(x_n)d \leq 0$$

Update $x_{n+1} = x_n + d$. Multipliers are updated based on the solution. Note at each step we use the active set and Newtonian methods.

- Using both gradient and Hessian info, along with quadratic approximation makes convergence near the optimum very fast
- Prone to converging to local minimum – choose x_0 wisely

Penalty Function Method

Idea: Add an extra term that penalizes any constraint violation

A classic is the quadratic penalty function given penalty parameter γ:

$$\min_x f(x) + \gamma \frac{1}{2} \left(\sum_{i=1}^{m} g_i(x)^2 + \sum_{j=1}^{k} \max\{0, h_j(x)\}^2 \right)$$

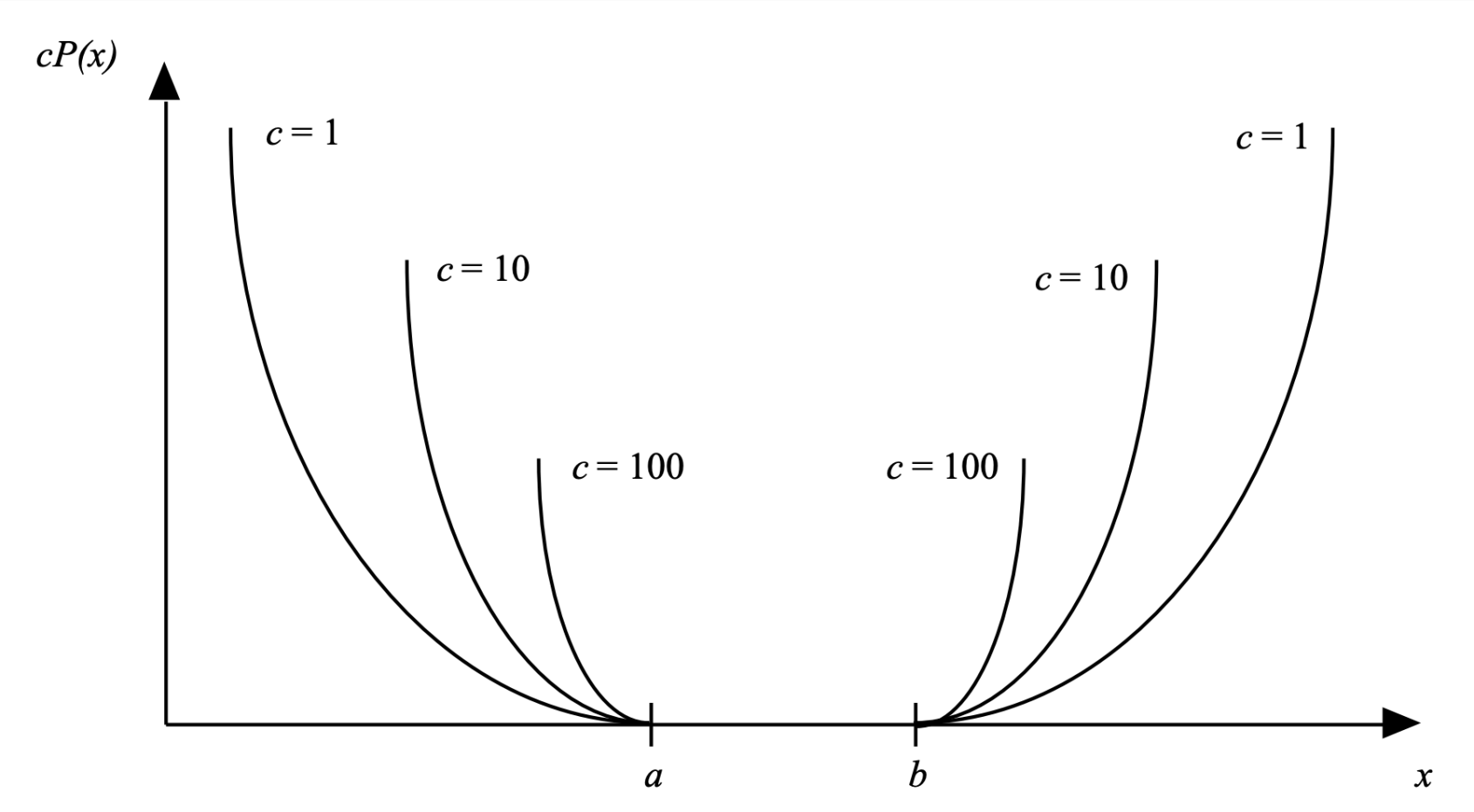
How should γ be chosen? Note that the original problem $\equiv \gamma \to \infty$
Sequential unconstrained minimization technique:

1. Choose an initial point x_0 that's infeasible and an increasing sequence $\{\gamma_0, \gamma_1, ...\}$
2. Optimize with penalty functions for violated constraints and γ_0 to get solution x_1. Use x_0 as the starting value in the optimization.
3. Iterate until $\|x_{n+1} - x_n\| < \epsilon$ for some threshold $\epsilon > 0$

This is an **exterior point method** which searches the infeasible space until the final solution. Be aware of scaling the constraints!

Penalty Function Method

Note the steep penalty when c is large (possible ill-conditioned gradient)



Source: Lluís Alsedà

Barrier Function

Idea: Opposite to penalty functions, use a barrier function to prevent solutions from leaving the feasible set

$$\min_x f(x) + \underbrace{\omega}_{\text{penalty parameter}} \sum_{j=1}^{k} \overbrace{b(h_j(x))}^{\text{barrier function}}$$

where common barrier functions include:

$$\text{Logarithmic Barrier: } b(h_j(x)) = -\ln(-h_j(x))$$

$$\text{Inverse Barrier: } b(h_j(x)) = -\frac{1}{h_j(x)}$$

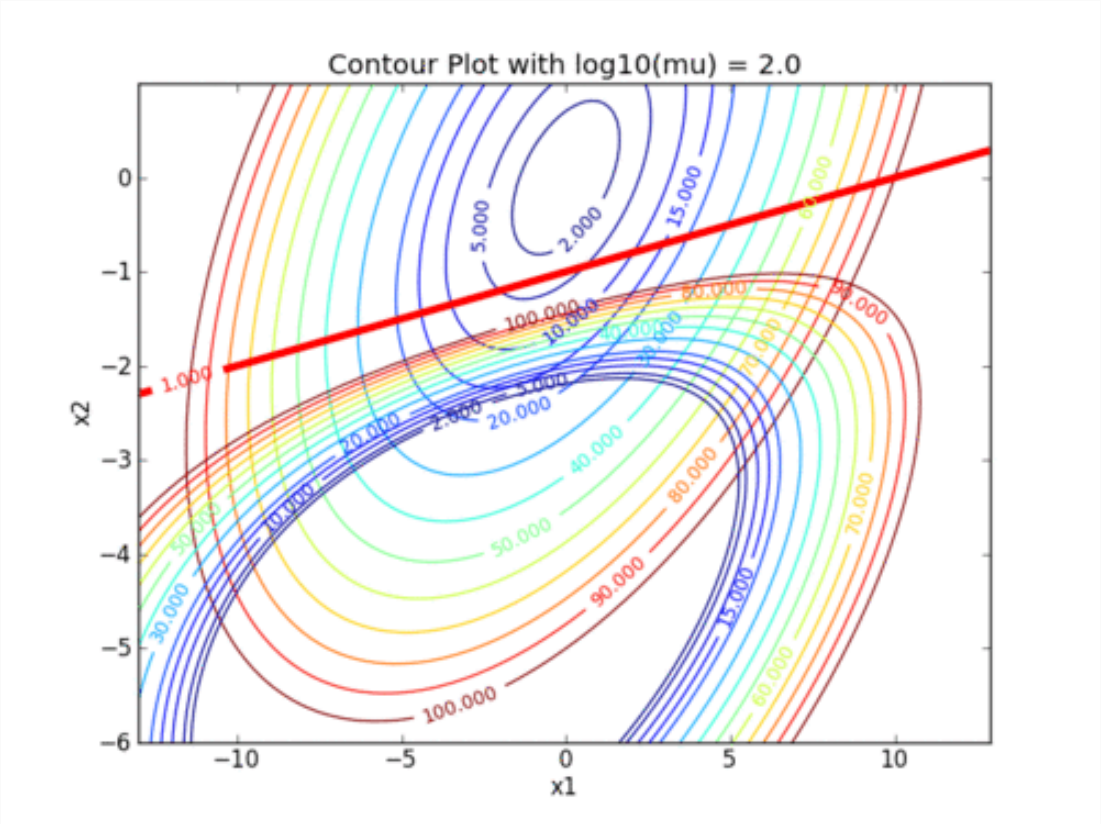
As $h_j(x) \leq 0$ approaches the boundary from inside, $b(h_j(x)) \to \infty$

1. Choose an initial starting point x_0 that's *feasible* (this is non-trivial) and a *decreasing* sequence $\{\omega_0, \omega_1, ...\}$ Optimize with x_0 and ω_0.
2. Iterate until $\|x_{n+1} - x_n\| < \epsilon$ for some threshold $\epsilon > 0$.

This is an **interior point method**. Equality constraints are hard!

Barrier Function in Action

In practice, barrier parameter is always positive



Source: APMonitor.com

Linear Programming

A special case arises when objective and constraints are all linear

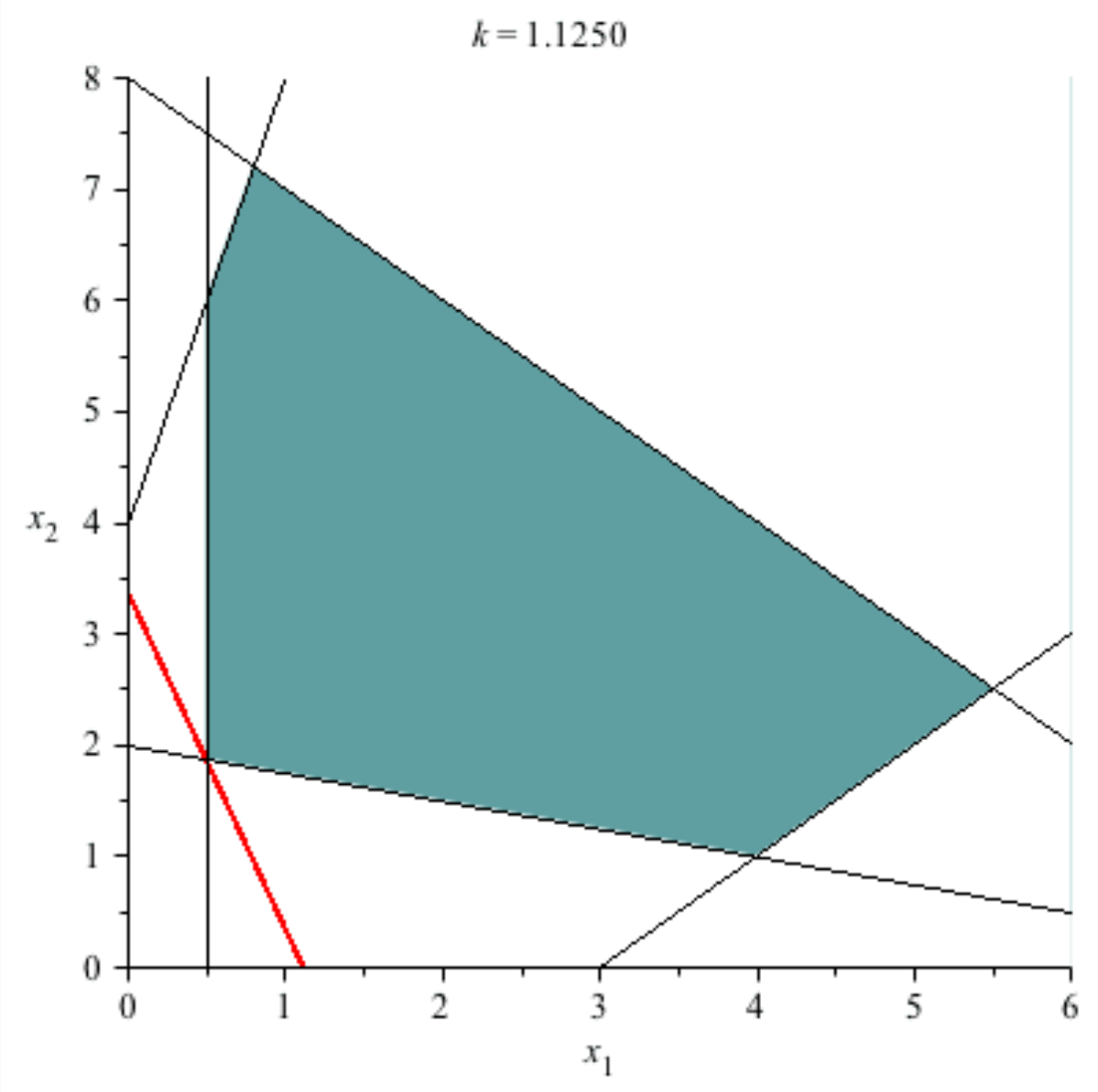
$$\min_x \quad C^T x \quad s.t. \quad Ax - b = 0, \quad x \geq 0$$

The simplex method is derivative-free:

1. Find some initial value x_0 that is a vertex of the simplex of constraints
2. Identify which vertex close by reduces the objective function more, go to the next point x_1
3. Traverse along the simplex vertices until the objective cannot be reduced further

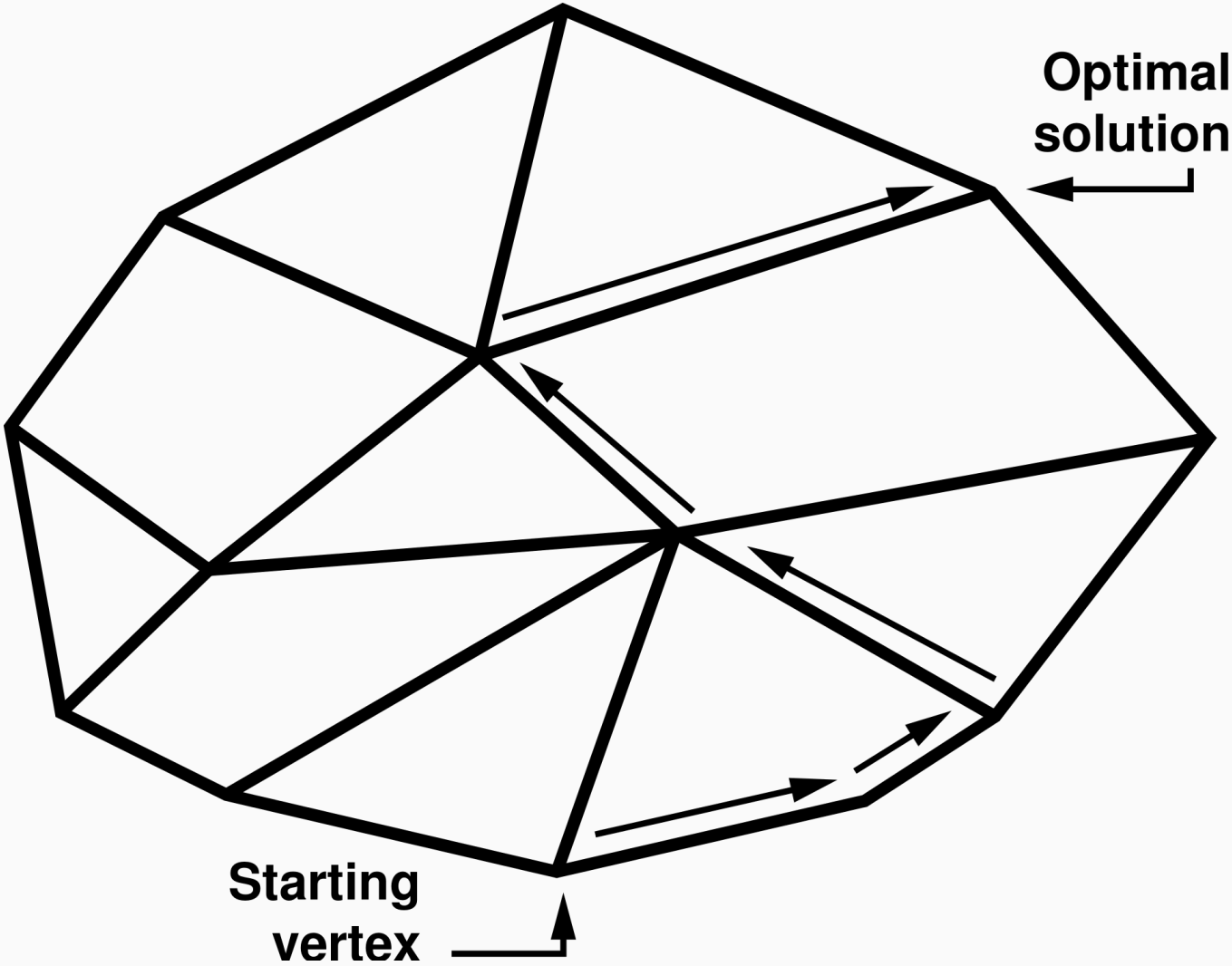
There's a finite set of vertices so generally fast, analytically solved via the tableau method.

Linear Programming in Action



Source

Simplex Algorithm



Source: Wikipedia

Constrained Optimization BY Linear Approximation

Idea: the derivative is unavailable, so linearly approximate the objective and constraints to solve successive linear problems instead

1. Initialize a starting guess x_0 and a simplex around the point ($n+1$ points for n dimensions)
2. Calculate the objective and constraints at all simplex vertices, and construct a linear approximation
3. Solve the linearized problem within the specified **trust-region**[2] to get x_1 (e.g. using Nelder-Mead)
4. Update the worst-performing simplex vertex with x_1 if the performance is better
5. Re-iterate the linear approximations with the new simplex until solution convergence

[2] The area around a point where the approximations are trustworthy. The radius is determined by a ratio between the predicted and actual reduction in $f(x)$.

Summary & Implementation

Summary of Constrained Optimization

Derivatives are ubiquitous – including of the constraints since they are part of Lagrangian. Increases computational complexity.

Method	Pros	Cons
SQP	Highly effective for smooth functions. Handles equality and inequality constraints.	Requires derivatives of the objective function and constraints.
Penalty	Converts constrained problems into unconstrained ones. Handles equality and inequality constraints.	Requires careful tuning of penalty parameters (especially when large), and derivatives.
Barrier	Particularly effective for inequality constraints. Maintains solution feasiblity.	Requires the initial point to be feasible, careful tuning, and derivatives.
COBYLA	Does not require derivatives, suitable for noisy functions. Handles equality and inequality constraints.	Convergence may be slower than gradient-based methods and for large-scale problems.

Minimization in Python

`scipy.optimize.minimize()` contains Python's minimizers, most come with the option of constraints:

- General:
  - `minimize(method='Newton-CG')`
  - `minimize(method='L-BFGS-B')`
  - `minimize(method='Nelder-Mead')`
- Constrained:
  - `minimize(method='trust-constr')`
    - Only equality constraints: use SQP with trust-region
    - Inequality constraints: use interior point method (i.e. barrier functions) with trust-region
  - `minimize(method='SLSQP')` (a type of SQP method)
  - `minimize(method='COBYLA')`